PRIFYSGOL ABERYSTWYTH UNIVERSITY

Aberystwyth University

Clann: investigating phylogenetic information through supertree analyses

Creevey, C. J.; McInerney, J. O.

Published in:
Bioinformatics

DOI:
10.1093/bioinformatics/bti020

Publication date:
2005

Citation for published version (APA):
Creevey, C. J., & McInerney, J. O. (2005). Clann: investigating phylogenetic information through supertree analyses. *Bioinformatics*, *21*(3), 390-392. https://doi.org/10.1093/bioinformatics/bti020

General rights
Copyright and moral rights for the publications made accessible in the Aberystwyth Research Portal (the Institutional Repository) are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the Aberystwyth Research Portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the Aberystwyth Research Portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

tel: +44 1970 62 2400
email: is@aber.ac.uk

Download date: 05. May. 2023

Clann Manual (version 3.0.0)

(Revised Manual released 16th November 2004)

Clann: Construction of Supertrees and exploration of phylogenomic information from partially overlapping datasets.

Homepage: http://bioinf.may.ie/software/clann/

Copyright © Christopher Creevey 2003-2005.

Clann was written by Chris Creevey while working in the lab of James McInerney at the National University of Ireland between 2003 and 2005.

Table of Contents:

AIM: 3
LEGAL STUFF: 3
DISCLAIMER: 3
BUG REPORTING: 4
REFERENCING CLANN: 5
BACKGROUND. 6
OPTIMALITY CRITERIA. 9
1. Matrix Representation using Parsimony (MRP) 9
2. Most Similar Supertree Method (dfit) 11
3. Maximum Quartet fit. (qfit) 16
4. Maximum Split Fit (sfit). 17
5. Average consensus (avcon). 18
OPERATIONS ON DATA. 19
Heuristic searches of tree-space: 19
Bootstrapping source trees: 21
YAPTP Test 21
INSTALLING AND RUNNING CLANN 23
THE CLANN INTERFACE. 25
Execute (or exe) command: 29
alltrees command: 35
hs command: 38
usertrees command: 44
bootstrap (or boot) command: 46
yaptp command: 51
includetrees command: 55
excludetrees command: 57
showtrees command: 59
consensus command: 62
deletetaxa command: 66
generatetrees command: 67
nj command: 71
rfdists command: 72
set command: 73
! command: 74
quit command: 74
TUTORIAL: 75

Aim:

To construct supertrees and explore the underlying phylogenomic information from partially overlapping datasets.
The program Clann has been developed to provide implementations of several supertree methods. The methods implemented all allow the investigation of data in a phylogenomic context. It is important that the user understands the advantages and limitations of these methods. It is also important for the user to know that the software is designed to perform a number of different tasks, however the interpretation of the results is left entirely to the user.

Legal Stuff:

Clann 3.0.0 © 2003-2005 Chris Creevey.
The software and its accompanying documentation are provided as is, without guarantee of support or maintenance. The whole package is licensed under the GNU public license, except for the parts indicated in the sources where the copyright of the authors does not apply. Please see http://www.opensource.org/licenses/gpl-license.html for details.

Disclaimer:

While we try to ensure that the software is free of bugs, this cannot be guaranteed. The software is provided as-is, with no guarantee that it will do anything, that it is suitable for any purpose whatsoever and that it will be of any use to anybody. We cannot be held responsible for any errors and we cannot be held responsible for the user being misled by any results they obtain when using the software. But that said, we think it works really well!

Bug reporting:

It is the intention of the authors to continuously develop and up-grade this software. Much of this development can only happen with user feed-back. With this in mind we will put all constructive comments into a "wish list" of upgrades and bug-fixes. If you are going to report a bug, we ask that you provide the following:

1) The version of Clann you are running.
2) The operating system you are using in as much detail as possible (like Mac OS 10.2.8).
3) A description of the exact nature of the problem (i.e. the operation that causes the crash or the repetition that the crash occurred).
4) A copy of the data that caused the failure. Without this it will not be possible to re-create the problem and fix it. Please note that all data provided will be treated with the up-most confidentiality.
5) Email all these details to Chris Creevey, (chris.creevey@gmail.ie)

Referencing Clann:

Clann has been published in Bioinformatics in 2005 under the following title:

Creevey C.J. and McInerney J.O. 2005 Clann: investigating phylogenetic information through supertree analyses. ***Bioinformatics*** 21(3): 390-2.

The Bootstrapping and YAPTP methods and the DFIT (most similar supertree algorithm) have all been described in the paper:

Creevey C.J., Fitzpatrick, D.A., Philip, G.A., Kinsella, R.J., O'Connell M.J., Travers, S.A, Wilkinson M. and McInerney, J.O. 2004 Does a tree-like phylogeny only exist at the tips in the prokaryotes? ***Proceedings of the Royal Society London*****, B series: Biological Sciences** 271 (1557): 2551-8.

Either or both of these publications should be cited if you use Clann in published work.

Other places we have used Clann:

Fitzpatrick, D.A., Creevey, C.J. and McInerney, J.O. (2005). Genome Phylogenies Indicate a Meaningful -Proteobacterial Phylogeny and Support A Grouping of the Mitochondria With the Rickettsiales. **Molecular Biology and Evolution** doi:10.1093/molbev/msj009.

Philip, G.K., Creevey, C.J. and McInerney, J.O. (2005). The Opisthokonta and the Ecdysozoa may not be Clades: Stronger Support for the Grouping of Plant and Animal than for Animal and Fungi and Stronger Support for the Coelomata than Ecdysozoa. **Molecular Biology and Evolution** 22, 1175-1184.

Wilkinson, M., Cotton, J., Creevey, C.J., Eulenstein O., Harris, S.R., Lapointe, F-J. McInerney, J.O., Pisani, D and J. Thorley (2005). The Shape of Supertrees to Come: Input Tree Shape Biases and Some Axiomatic Properties of Fourteen Supertree Methods. **Systematic Biology** 54(3):419-31.

Background.

One way to build larger more comprehensive phylogenies is to combine the vast amount of phylogenetic information already available. A supertree does this by combining all the taxa from a collection of fundamental (or source) trees into a single phylogeny (*1*). An ideal supertree that agrees completely with all its source trees is called a strict supertree, and can only result when all its source trees are compatible. Two source trees are compatible if, when only their shared taxa are considered, their relationships to each other are the same in both trees (*1*). However an ideal strict supertree is rarely found because phylogenies based on different genes are subject to different evolutionary processes and because of events like lateral gene transfer and duplication.

With the availability of whole genomes, supertree methods have become very important in reconstructing whole genome phylogenies (*2*). New methods (*2-6*) have been developed, each approaching the problem from a different perspective.

Methods based on gene content (*3, 7, 8*) build phylogenies based on the presence or absence of particular genes or families of genes. The concept being that closely related species should share a large proportion of genes, while distantly related species should either have lost genes since they last shared a common ancestor or never have had many genes in common in the first place. Ideally this gene loss would be constant over time and would also be a good estimate of how long its been since they last shared a common ancestor (*3, 7, 8*). However, selective pressures acting on gene loss (as in adaptation to endosymbiosis (*9*)) or gene acquisition by horizontal gene transfer (as in adaptation to extreme environments

(*10*)) or gene preservation (for functionally important genes (*11*)) may alter this rate.

Analysing gene order, which follows the same idea as methods based on gene content, is another way of comparing genomes. The concept here is that those genomes that are more closely related have undergone less genomic rearrangement than those that are more distantly related (*4*). Studies of gene order (*12*) have successfully identified pairs of genes that are conserved in their proximity in several genomes, and the presence-absence matrices of these pairs have been used to construct supertrees. Problems which confound this approach include the effects of selective pressures and increased chances of lateral gene transfer because operons in genome rearrangement are generally transferred as one piece (*13*).

Another way of reconstructing genome phylogenies is to calculate the evolutionary distances between pairs of orthologs. Grishin et al. (*14*) and Wolf et al. (*5*) used different methods of identifying orthologs and calculating the evolutionary distances between them. They both examined prokaryotic phylogenies, and used the distances calculated to reconstruct the supertree. The different approaches to using this method show a wide distribution of results, probably because of lateral gene transfer, misidentification of orthologs and sampling error (*15*). calculating the evolutionary distances between them. They both examined prokaryotic phylogenies, and used the distances calculated to reconstruct the supertree. The different approaches to using this method show a wide distribution of results, probably because of lateral gene transfer, misidentification of orthologs and sampling error (*15*).

Concatenated sequence alignments have proven to be a very useful method of reconstructing genome phylogenies. In this method, orthologs are identified in the genomes in question and their associated sequences are aligned. Then all the orthologs for each genome are concatenated (in the same order) and various standard phylogeny reconstruction techniques are then used on the concatenated genome sequences (*16-20*). There are limitations to this approach however. All the concatenated alignments must include the same set of species, and forcing them into one alignment, assumes that the same process of evolution has been acting on all the genes. Brown (*19*) tried to overcome this difficulty by choosing genes that had a function during translation, this however limited the dataset to 23 orthologous proteins from only 45 species of prokaryotes. Besides these limitations, concatenated alignments have produced genome phylogenies with good resolution (*16, 19*), however the usable datasets have been rather small.

A final method of constructing genome phylogenies is that of combining the information of multiple trees. This allows us to construct phylogenies for individual orthologous genes, and to combine them into an overall supertree. Baum (*21*) and Ragan (*22*) (MRP) coding schemes have been used for this purpose (*2*) as well as calculation of nearest neighbours across many source trees (*23*), combination of weighted trees (*24*) and methods that use maximum likelihood analysis of quartets (*25*). This approach suffers from the problem of reconciling the information from many trees, filtering out the homoplasy and the possibility of including paralogs in the analysis. However it does allow for large proportions of each genome to be included in the analysis, and probably for that reason better reflects the "true" genomic phylogeny (*15*).

Optimality criteria.

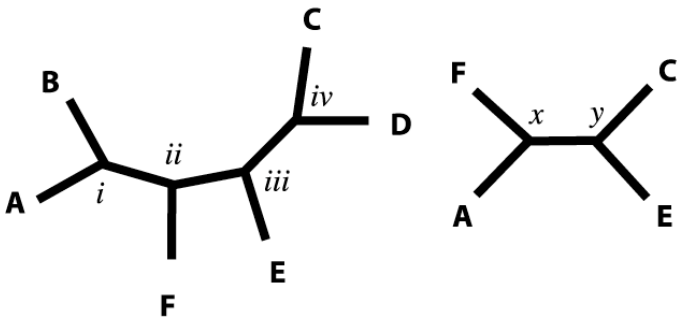
At present there are 4 supertree methods implemented in Clann that combine information from multiple (non-overlapping) trees. These methods are used as optimality criteria for searching tree-space.

1. Matrix representation using Parsimony (MRP) (using PAUP*)
2. Most Similar Supertree (dfit)
3. Maximum Quartet fit (qfit)
4. Maximum Splits fit (sfit)
5. Average Consensus (avcon)

1. Matrix Representation using Parsimony (MRP)

Matrix representation using parsimony (MRP) creates a matrix whose characters refer to the topologies of the source trees. This method of supertree construction was proposed by Loomis and Smith (*26*), and later refined by Baum (*21*), and Ragan (*22*). The method works by examining each internal branch of each rooted source tree, and assigning a '1' to any taxa contained within the clade defined by that internal branch. A '0' is assigned to any taxon that is contained within the source tree, but not in the clade, and a '?' is assigned to any taxa not present in the source tree (figure 1). The columns in the matrix each represent one internal branch in one of the source trees, so the total number of columns in the matrix is equal to the total number of internal branches across all the source trees. The matrix is then analysed using parsimony, and the supertree phylogeny is reconstructed. Strict supertree construction is conservative however and the result cannot conflict

with any phylogenetic relationships in any source tree. If MRP is used with source trees that are not consistent, the reconstructed phylogeny will contain polytomies reflecting the relationships in which the source trees conflict. Such conflict among source trees may be common, becoming more likely as the numbers of source trees or shared taxa increases (*1*). MRP handles conflict by weighing the evidence in different source trees, without any tree having a power of veto. Incompatible source trees can also be analysed using MRP by either treating the conflicts with consensus methods or selective removal of problematic shared taxa until the remaining sub trees are compatible (*1, 27*). MRP is also sensitive to the size of the source trees used, with smaller trees seeming to give a better result (*28*). However, the subsequent loss of coverage can be problematic (*29, 30*).



Taxa	*i*	*ii*	*iii*	*iv*	.....	*x*	*y*
A	1	1	0	0	...	1	0...
B	1	1	0	0	...	?	? ...
C	0	0	1	1	...	0	1 ...
D	0	0	1	1	...	?	? ...
E	0	0	1	0	...	0	1 ...
F	0	1	0	0	...	1	0 ...

Figure 1: Construction of supertrees by matrix representation. A column represents each internal node of each source tree. If a taxon is contained within the clade defined by that internal branch a '1' is placed beside it in the column. If the taxon is not in the clade

but is on the tree a '0' is placed beside it in the column other wise if the taxon is not on the tree a '?' is placed beside it in the column. The matrix is then analysed using parsimony and the most parsimonious phylogeny is reconstructed (*21, 22*). Bootstrap values may be incorporated into the matrix by representing a column repeatedly proportional to the bootstrap value at the node that it represents (*2*).

2. Most Similar Supertree Method (dfit)

This scoring method compares each source tree separately to the supertree by comparing the distance matrix derived from a source tree to a distance matrix derived from the supertree. The differences between the matrices are scored and the sum of the scores from all the comparisons is calculated. This sum is the score assigned to the supertree.

The comparison of a supertree to a source tree is only possible if the two trees are the same size. However most source trees have fewer taxa than the supertree. How then is a comparison of two differently sized trees possible? For each comparison the supertree is reduced to the same size as the source tree. This is achieved by pruning those taxa from the supertree that are not found in the source tree (Figure 2). Once the pruning is complete, a distance matrix is calculated for the supertree by counting the number of nodes separating each taxon from every other taxa (Figure 3). This distance matrix is the same size and comparable to the distance matrix from the source tree.

The differences between the source tree distance matrix and the supertree distance matrix is calculated as follows. As both of these matrices contain information on distances between the same taxa,

it is possible to check if the distance between any two taxa differs in either matrix. The absolute value of the difference between these distances is calculated (Table 1). The sum of all these absolute values arising from the comparison between the two matrices is then calculated. This score is a measure of the discrepancy between the source tree and the supertree being examined. If the pruned supertree and the source tree are the same, then the score is zero.

However there is a correlation between the size of the tree score and the size of the source tree being examined. Large trees result in large scores and small trees result in small scores. To normalise against this effect, the sum of the absolute differences can be divided by the total number of comparisons in the matrix. The user must decide whether a normalisation like this (where every tree has the same vote in the result) is true to the peculiarities of their data. The user can choose to not normalise, thus allowing larger trees to have a bigger "vote".

It is advisable to try several different normalisation techniques on any data analysed as this can reveal whether or not any results obtained are biased towards large or small trees.

As an example, in Table 1 there are 5 taxa, this means that there are 15 possible comparisons between each of these. This is a Table of the absolute differences between the two matrices in Figure 3.3. The sum of all the cells in Table 3.1 is 20 and is the sum of the absolute differences. This is normalised by dividing it by the total number of comparisons, so the normalised score is 20 divided by 15, which is 1.333.

The total score for any given supertree is therefore given by the sum of the normalised scores for every source to supertree

comparison. This represents the similarity of the supertree to the source trees. Numerous other tree-to-tree distance or fit measures could be used to define optimal supertrees (*31*). The present method is most similar to the Average Consensus procedure with branch lengths all set at unity (*32*).

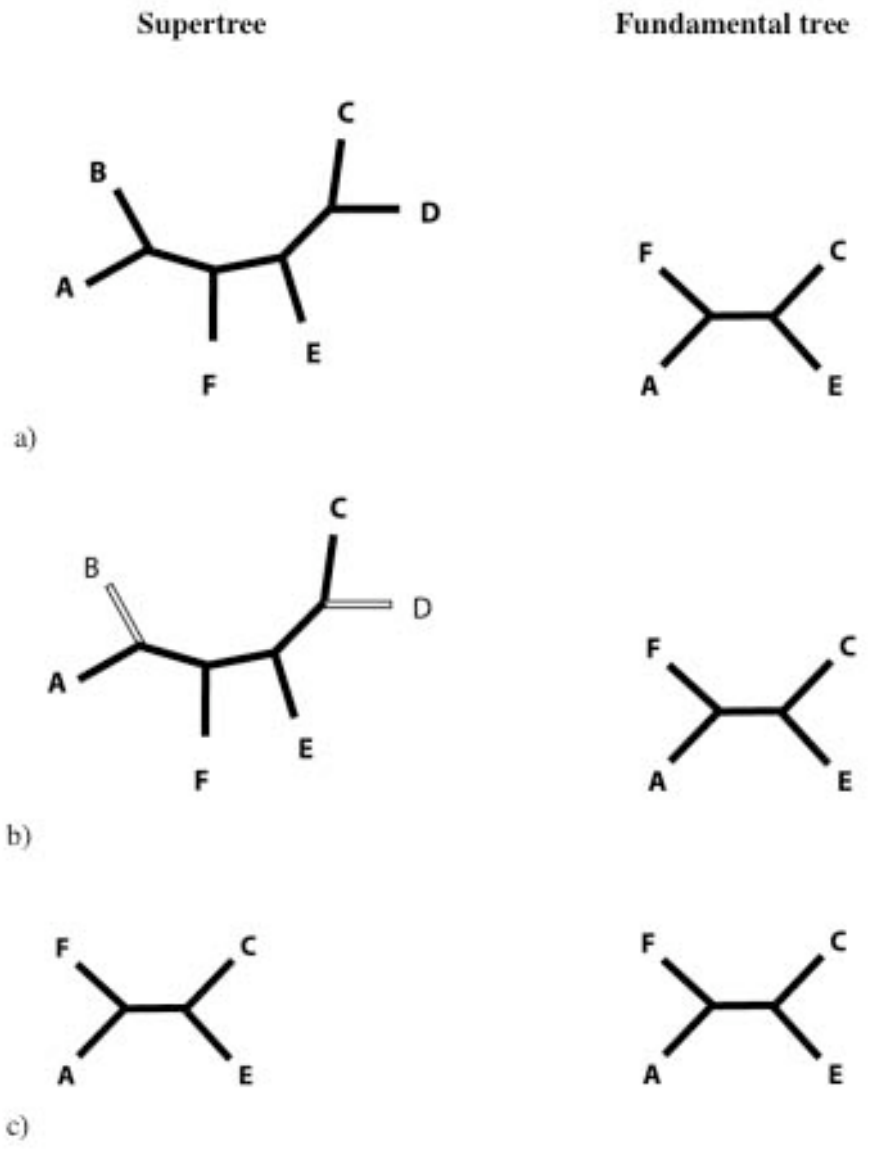
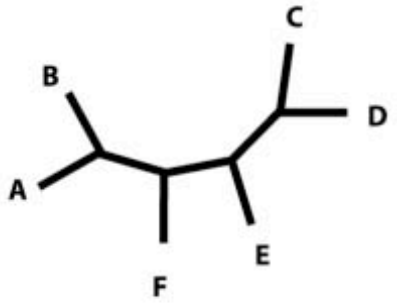
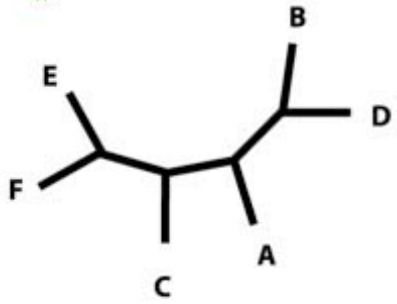


Figure 2: Supertree pruning. a) The supertree contains more taxa than the source tree, so it is necessary to identify those taxa and internal branches that are not part of the source tree, shown in white in b). As can be seen in c) these taxa and branches are then pruned leaving the supertree the same size as the source tree.



(a)

	A	B	C	D	E
B	1				
C	4	4			
D	4	4	1		
E	3	3	2	2	
F	2	2	3	3	2



(b)

	A	B	C	D	E
B	2				
C	2	3			
D	2	1	3		
E	3	4	2	4	
F	3	4	2	4	1

Figure 3: Calculating distance matrices for unrooted trees. The numbers in the Tables represent the number of nodes separating any two taxa. Note the differences in tree (a) and (b) and how it changes the respective distance matrix.

	A	B	C	D	E
B	1				
C	2	1			
D	2	3	2		
E	0	1	0	2	
F	1	2	1	1	1

Table 1: The absolute difference of the matrices in Figure 2. The sum of these differences gives the score for the similarity between the two phylogenies, in this case 20. If the two phylogenies were identical this score would be 0.

Algorithmic procedure:

The algorithm proceeds as follows:
Calculate a distance matrix for every source tree in memory.
For every supertree in the supertree file (or for each supertree created during the heuristic search or exhaustive search algorithm)

1) Build the supertree in memory
2) For each comparison to a source tree:
    i. Prune the supertree to the same leaf set as the source tree
    ii. Calculate a distance matrix for the pruned supertree.
    iii. Calculate the absolute sum of the differences between the distance matrix of the pruned supertree and the distance matrix of the source tree.
    iv. Restore the supertree to its original size.

3) Sum the scores from all the comparisons to the source trees. This is the score assigned to the supertree.
4) Dismantle the supertree in memory.

Repeat until all the supertrees have been examined (or the optimal tree has been found).

3. Maximum Quartet fit. (qfit)

In this method, all the quartets (relationships between any 4 of the taxa) from any proposed supertree are determined. All the quartets from the source trees are also determined. A score is then calculated which is defined by the number of quartets that are shared between the supertree and the set of source trees. This is used as an optimality criterion to determine the supertree that has the maximum quartet fit (i.e. shares the largest number of quartets with the source trees).

Issues that affect this optimality criterion include large tree bias. Large trees have very many more quartets than small trees. The number of quartets increases non-linearly to very large numbers very quickly. In an attempt to adjust for this effect the contribution of any quartet match to the overall score can be normalised in several ways. Firstly we can leave all quartets to contribute to the same degree, this is equivalent to stating that bigger trees have more information and so should be allowed to make a bigger contribution to the final answer. Secondly we can normalise the score that any quartet match contributes to the final score by dividing it by the number of quartets contained in that source tree. This is equivalent to stating that every tree should contribute equally, regardless of size, to the final score. Thirdly we can normalise the score to allow bigger trees have more say than smaller trees, but not to the extent if we didn't normalise the score at all. In this method, the contribution of any quartet is normalised by dividing it by the number of taxa minus three (n-3), on the source tree from which it came.

All three of these normalisations are included as part of the algorithm in Clann, and it is up to the user to decide which works

best with their data. It is advisable to try several different normalisation techniques on any data analysed as this can reveal whether or not any results obtained are biased towards large or small trees.

4. Maximum Split Fit (sfit).

In this method, all the splits (components) from any proposed supertree are determined. All the splits from the source trees are also determined. A score is then calculated which is defined by the number of splits that are shared between the supertree and the set of source trees. This is used as an optimality criterion to determine the supertree that has the maximum split fit (i.e. shares the largest number of splits with the source trees). This method allows missing data to be considered as not conflicting, this then allows the comparison of splits from differing sized trees.

Once again, large tree bias can also affect this optimality criterion. To try to take this into account Clann implements two options. Firstly the user can choose to allow the contribution to the final score of each matched spit to be equal regardless of the size of the source tree that the split came from. This is equivalent to allowing larger trees to have more of a say in the result. Secondly, the user can choose to normalise the contribution of any matched split by dividing by the number of splits. This has the effect of giving each tree an equal "vote" in the final result, regardless of the size of the tree.

Again it is advisable to try several different normalisation techniques on any data analysed as this can reveal whether or not any results obtained are biased towards large or small trees.

5. Average consensus (avcon).

The average consensus method (*24, 33*) combines the information from multiple trees by calculating the path length from every taxa to every other taxa on each of the source trees. This method utilizes branch lengths (if present) or assumes a branch length of unity (a branch length of 1) if not present. The resulting supertree from this method will have branch lengths.

The specifics of the method are as follows:

- The distance from each taxon to every other taxon on a tree is calculated. This is the equivalent of the sum of the branch lengths in the path between the two taxa.
- The average distance of each taxa to every other taxa across all the source trees is calculated.
- In the supertree context, some taxa may never appear together on an input tree. In this case we would not have a value for the distance between these two taxa. To remedy this, the value of the missing cells is estimated using either ultrametric or 4-point-condition estimates (*34*). This results in a distance matrix with distances from each taxa to every other taxa.
- A least squares method is used to estimate a supetree phylogeny (with branch lengths) that best describes the distance matrix. (this part is carries out by PAUP* in Clann) You also have the option of doing a neighbor joining supertree from the calculated distance matrix (see the command 'nj').

Operations on Data.

For each of the optimality criteria implemented, several different methods of analysing the phylogenomic content are implemented in Clann. These methods include exhaustive searches of tree-space, heuristic methods of searching tree-space, methods of bootstrapping (with replacement) the trees to examine the underlying support for any hypothesis and methods for determining whether any phylogenomic signal present in the data is better than what would be expected from random data.

Heuristic searches of tree-space:

This leads to the problem of where a supertree comes from. Given 20 taxa there are over $8x10^{21}$ different possible phylogenies. How then can the correct phylogeny be found? One method is to do an exhaustive search of tree space for the correct phylogeny, however this is not feasible for anything over 12 taxa datasets as it would take too long to complete on a single computer. Another method is to use a heuristic search of tree space. Heuristic searches use various methods to only search that part of tree space where the correct tree is likely to be located.

The heuristic methods used in the software are blind hill-walking algorithms. Heuristic hill-walking algorithms try to find the optimum answer to a problem without trying all possible answers. Initially an answer is proposed and is assessed. A modification is then made and this new answer is assessed. If the new answer is better, then it is retained and used as the starting point to find the next best answer. Otherwise, the new answer is discarded and the previous answer is used as a starting point. Traditionally this is viewed as

climbing a hill where we are trying to find the top of the hill (the best answer). Any step that brings us downhill is not taken, and any step that brings us uphill is taken. As this is a blind hill-walking algorithm it is not possible to know the surrounding topology and when the top of a hill is reached it is not possible to ascertain if this is the highest hill in the landscape or simply a "local optimum". This is makes it difficult to have any confidence in the result of a single heuristic hill-walk and usually it is necessary to perform the hill-walk several times using different starting points to make sure that the same answer is found every (or most of) the time.

The two heuristic algorithms implemented in Clann are nearest neighbor interchange (NNI) and sub-tree pruning and re-grafting (SPR) as described and implemented in PAUP* (*35*).

The nearest neighbor interchange algorithm adjusts trees by swapping the position of two neighboring branches. If this swap improves the tree then that adjustment is retained, otherwise the branches are swapped back. A strict NNI only swaps two branches if they are neighbours, that is if they are only a distance of one node away from each other. However the algorithm may be expanded to allow branches to be swapped that are a distance of several nodes away from each other. An upper limit can be enforced to the distance separating two branches that may be swapped. This method of tree searching is considered quite gentle, as it cannot change the topology of the tree being examined, but merely change the position of the branches on the tree. Any two branches of a tree may be swapped (including an internal branch for an external branch) as long as one branch is not a descendant of the other. When an internal branch is moved, the entire clade that it defines (everything below it) also moves.

The second heuristic algorithm implemented is called sub-tree pruning and re-grafting (SPR). This method removes a section of the tree and re-grafts it back onto the tree in a different position. If an internal branch is moved, then the clade that is defined by that internal branch is moved also. This method searches tree space far more rigorously than NNI as the whole topology of the tree is allowed to change. There are few limitations to this search algorithm but one is that the formation of polytomies in the phylogeny is not allowed. This is because of the possibility of resulting in a completely unresolved tree. However unlike NNI, SPR has the capability of breaking up and resolving polytomies.

Bootstrapping source trees:

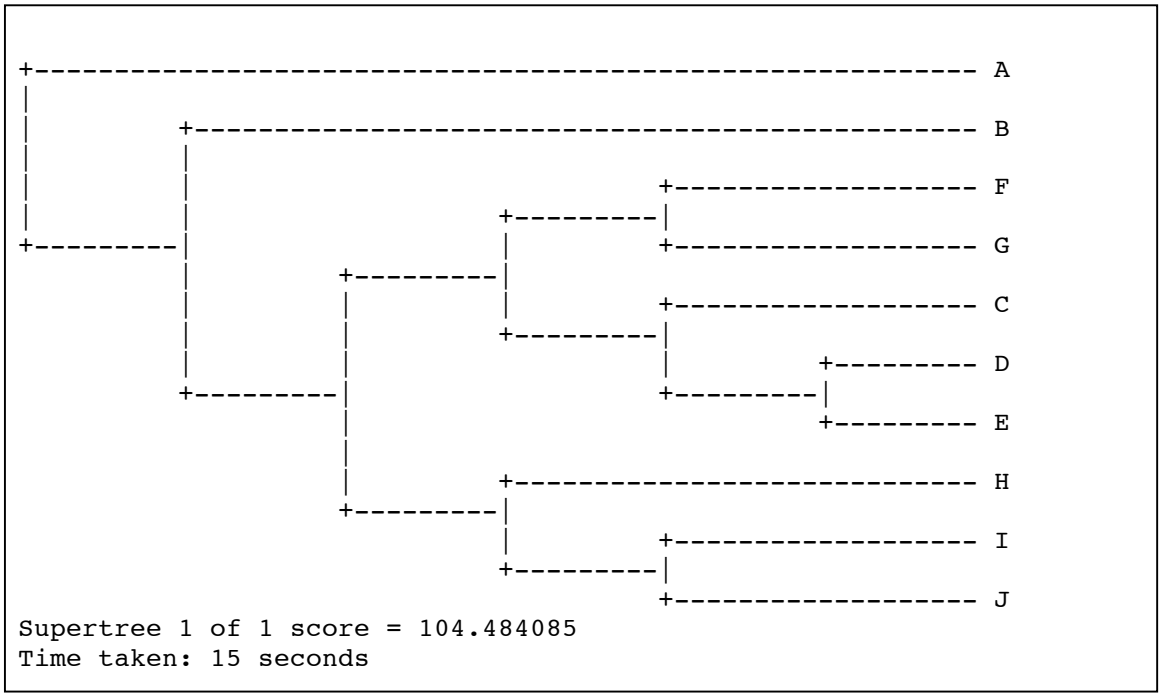
The source trees represent one possible set of trees that could have been used in the analysis. Choosing a slightly different set of source trees may result in a different optimal supertree. In order to estimate the likely nature of the universe of optimal supertrees, the source trees may be bootstrapped. For each bootstrap replicate, the source trees are sampled with replacement until a new dataset is created with the same number of source tree scores as the original dataset. This means that some source trees may be represented in the data set more than once, while others may not be represented at all. For each repetition, the supertree that best represents this (bootstrapped) set of source trees is determined. Repeating this procedure a large number of times gives an indication as to how much support there is for any supertree phylogeny.

YAPTP Test

A randomisation method to test the null hypothesis that the phylogenetic signal in the gene trees was no better than random is also implemented in Clann. This is called a YAPTP (Yet Another

Permutation Tail Probability) test. For each gene tree, we removed the taxon names and randomly reassigned them to the leaves. This removes any congruent phylogenetic signal between gene trees, while leaving the numbers, sizes and shapes of gene trees, the frequency with which any particular taxon was found across the gene trees, and the frequency of co-occurrence of any group of taxa within gene trees unaltered. A search of tree space can then be carried out and the score of the best supertree recorded. The user can repeat this test as many times as required and the distribution of the resulting scores can be compared to the score of the real data (or the distribution of scores from bootstrapping) to check that the real data contains a signal that is better than random.

Clann will display the results of an exhaustive or heuristic search graphically to the screen (as in the box below) but will also print the tree(s) to a file in nested parenthesis format (for viewing in programs such as "treeview" [http://taxonomy.zoology.gla.ac.uk/rod/treeview.html]) and also graphically in Postscript format to the file "supertree.ps". The file "supertree.ps" will be overwritten after every search.



Installing and running Clann

Clann is available to download at the website of the Bioinformatics and Pharmacogenomics laboratory at NUI Maynooth, Ireland.
http://bioinf.may.ie/software/clann

At the moment there are binary executables for four different Operating systems:

Apple Macintosh OS X (10.x)

Redhat Linux

Microsoft Windows PC.

Downloading:
When you click on the link for whatever operating system you are using, the appropriate version of Clann is saved to your computer's hard-drive. This is compressed to increase the download speed. In Microsoft Windows, and both Apple operating systems this file should be automatically uncompressed leaving you with the program ready to run. However, if this does not occur in Microsoft windows use the program 'winzip' (http://www.winzip.com) to uncompress Clann, or in either Apple Macintosh operating system use 'stuffit-expander' (http://www.aladdinsys.com).
In Redhat linux, issue the command "*gunzip clann.tar.gz*" to uncompress the executable. Follow that command with *"tar –xvf clann.tar"* to complete the installation.

Note:
The versions of Clann for Redhat linux and Mac OS X (10.x) are command line programs. To run Clann in either of these operating systems it is necessary to use a terminal window or shell. It may also be necessary in these operating systems to change the

permissions to allow clann to be run for the first time. This can be achieved by typing the command "*chmod a+x clann*".

Installing Clann:

In Mac OSX an installation script is included which installs the readline and ncurses libraries (if needed) before putting Clann into the folder /usr/bin/. Once in this location (and the user starts a new terminal window) Clann will be visible to the operating system from any directory. To run the installation, simply double click the icon "install.command", any errors in the installation process will be reported to screen. An administrative password will be needed to successfully install Clann.

In Redhat linux, the Clann program should either be located in the same directory as the input files, or somewhere on your path (like *~/bin/* or */usr/local/bin*). If you do not know which directories are on your path, ask your systems administrator.

Running Clann:

In Microsoft windows double clicking on the icon associated with Clann will run the program. In these operating systems, the actual Clann program **must** be located in the same directory as the input files, an alias or shortcut to Clann will not suffice.

To run Clann in MacOSx or red-hat Linux, type the command "*./clann*" or "*clann*" in a terminal window.

The Clann interface.

Clann was designed to have a "Paup*-like" (*35*) interface. This type of interface uses user specified commands to run the various analyses unlike a menu driven program where you are given a specific set of options. It is hoped that this will enable the evolution of the software since it will be easy to add extra features and analyses.

When Clann is started the user is presented with a simple prompt:

```
               ***************************************************
               *                                                 *
               *                  Clann  3.0.0                   *
               *                                                 *
               * web:   http://bioinf.may.ie/software/clann      *
               * email: chris.creevey@gmail.com                  *
               *                                                 *
               *   Copyright Chris Creevey 2003-2005             *
               *                                                 *
               ***************************************************

clann>
```

This is the standard prompt, to which the program will return after completion of any command.

A list of commands is available by typing the command "help", this results in the following list:

```
Available Commands:
        ----------------------------

The following commands are always available:

    execute         help            quit            set             !

The following commands are only available when there are source trees
in memory:

        alltrees                excludetrees            rfdists
        consensus               generatetrees           showtrees
        bootstrap               hs                      usertrees
        deletetaxa              includetrees            yaptp
        nj

        ----------------------------

 type a command followed by '?' to get more detailed information
        for example: exe ?

clann>
```

This gives the user a list of the available commands.

For further information on any of these commands it is possible to type the name of the command followed by a "?". This will result in a detailed list of the options for this command. For example the commands "alltrees ?" can give the following result.

```
clann> alltrees ?

alltrees [options]

      Options     Settings                        Current
      ================================================

      range       <treenumber> - <treenumber>     *all
      savetrees   <filename>                      top_alltrees.txt
      create      yes | no                        *no
      weight      equal | comparisons             comparisons

                                              *Option is nonpersistent
      ================================================

clann>
```

This explains that the command "alltrees" may be followed by several options that affect the way in which the command runs. A user could issue the command:

"alltrees range 3 – 100 create weight=equal"

In this method, a command may be followed by as many options (in any order) as desired. The options (like create and range in above) and qualifiers (like 3, 10 and equal in above) only need to be separated by a valid separator. A valid separator may be a space, minus sign "-" or equals sign "=". The important thing to remember is that you follow any option with its correct qualifier, otherwise the command will not run. All the valid qualifiers for any option are listed beside the options in the above help display. There is such a help display for every available command (except "quit").

All commands and options must be typed in lower case, otherwise Clann will not recognize the commands.

When one of the commands is run, a summary of the settings used are printed to the screen. This is available for heuristic searches, bootstrapping and YAPTP tests for all criteria. The summary will lokk something like the following:

```
Heuristic Search settings:
        Criterion = Most Similar Supertree (dfit)
        Heuristic search algorithm = Sub-tree Pruning and Regrafting (SPR)
        Maximum Number of Steps (nsteps) = 5
        Maximum Number of Swaps (maxswaps) = 1000000
        Number of repetitions of Heuristic search = 10
        Weighting Scheme = comparisons
        Starting trees = Top 10 random trees chosen from 10000 random
samples
        Output file = Heuristic_result.txt

Progress Indicator:*
```

The progress indicator increments every 5 seconds. When it is a "*" the software is carrying out the random sampling of tree-space. When the indicator is a "=" the software is carrying out the heuristic searches of tree-space.

Note:
The available options for any command may change depending on the optimality criterion chosen. Users should check the available options before running any command.

The rest of this section will deal with each of the commands, one by one, explaining how to use each option and what will be the result of each command.

Execute (or exe) command:

Purpose:

To read in a file containing source trees for analysis.

Options:

the user must specify the filename after the word "execute" (or "exe"). For example:

"exe real.ph"

The one other option available to the user is 'maxnamelen'. This option allows the user to specify that the software should only look at the first x number of characters when reading the taxon name. This allows the inclusion of other information along with the name of the taxa.

This file may contain the source trees in phylip format, **with or without** branch lengths.

```
(A,B,(C,(D,E),((F,G),H)));
(B,A,((C,(E,D)),(G,F)),(I,H));
(B,(((E,D),C),A,(F,G)),((I,J),H));
(E:0.2,D:0.01,(H:0.01,(J:0.01,I:0.1):0.2):0.1);
(H,(B,((((E,D),C),(G,F)),A)),(I,J));
(D,E,(C,(A,(B,(H,I)))));
(A:0.1,((E:0.001,D:0.11):0.01,C:0.02):0.3,(F:0.1,G:0.01):0.01);
(G,F,(((C,(E,D)),B),A));
(J,I,(H,((((C,(D,E)),(F,G)),A),B)));
(D,E,(C,((F,G),B,(H,(I,J)))));
```

The input trees may also contain internal node labels like bootstrap proportions (clann will ignore these).

Weights for individual trees can also be included and clann will use these during the supertree analyses.

For instance:

```
(A,B,(C,(D,E))) [0.001];
(A,E,(F,G)) [2.1];
```

This results in the first tree having a weight of 0.001 and the second tree having a weight of 2.1. Note that the weights in the square brackes comes **before** the semi-colon. Anything in square brackets **after** the semi-colon will be considered a label (or name) for the tree.

For instance:

```
(A,B,(C,(D,E))) [0.001]; [Tree 1]
(A,E,(F,G)); [Tree 2]
```

Clann will also read nexus formatted input files.

In this format, clann will ignore any data blocks not intended for its use (like sequences) and will read trees from the file with or without a translation table.
You may also include further information such as names of trees or weights.
Finally, you may also include a 'clann block' at the end of the file. This should include any commands that you wish clann to perform on the data. [Clann will ignore other command blocks (like for PAUP* or MrBayes)].

For instance:

```
#NEXUS
Begin trees;  [Test Tree file in nexus format]
Translate
                1 taxon_A,
                2 taxon_B,
                3 taxon_C,
                4 taxon_D,
                5 taxon_E,
                6 taxon_F,
                7 taxon_G,
                8 taxon_H,
                9 taxon_I,
                10 taxon_J,
                11 taxon_K,
                12 taxon_L,
                13 taxon_M,
                14 taxon_N,
                15 taxon_O
                ;
tree PAUP_1 = [&U] [&W 1.5] (1,((((((2,7),8),(((11,(12,13)),15),14)),((3,4),(5,6))),10),9));
tree PAUP_2 =  [&W 2.1] (11,((((((2,7),8),(((1,(12,13)),15),14)),((3,4),(5,6))),10),9));
tree PAUP_3 = [&U] [&W 3] (2,((((((1,7),8),(((11,(12,13)),15),14)),((3,4),(5,6))),10),9));
tree PAUP_4 = [&U] [&W 0.001] (6,((((((2,7),8),(((11,(12,13)),15),14)),((3,4),(5,1))),10),9));
End;


Begin Clann;
set criterion=dfit;
hs nreps=1 sample=1;
showtrees;
quit;
endblock;
```

Result:

This command reads all the source trees into memory and performs some simple analyses, the results of which it prints to screen.

Firstly Clann will determine the number of unique taxa in the source trees and determine the size of supertree space for this dataset. These results are printed to screen:

```
Reading Newhampshire (Phylip) format source tree file

        Input File summary:

        --------------------------------------------------
                Number of input trees: 920
                Number of unique taxa: 10
                Total unrooted trees in Supertree space?
                        2.02702e+06
```

Next Clann determines the number of times any taxa appears in a tree. This may be helpful in determining any taxa that are poorly represented.

```
Occurrence summary:

          number  Taxa name           Occurrence

          0       A                   672
          1       B                   723
          2       C                   781
          3       D                   830
          4       E                   831
          5       F                   581
          6       G                   519
          7       H                   687
          8       I                   551
          9       J                   425
```

Next, Clann determines the co-occurrence of each of the taxa. That is, the number of times any two taxa appear together in a source tree. This may enable the user to determine where weak support for a relationship may be due to low co-occurrence of the taxa.

```
Co-occurrence summary:

         Taxa Number

              0     1     2     3     4     5     6     7     8     9
        0     -
        1     582   -
        2     602   628   -
        3     627   653   739   -
        4     622   652   744   806   -
        5     484   496   541   551   552   -
        6     442   452   483   493   497   496   -
        7     518   588   577   617   613   444   402   -
        8     425   462   456   491   489   376   348   504   -
        9     355   376   366   383   379   332   318   390   408   -
 ---------------------------------------------------
```

Finally, Clann summarises the input file according to the sizes of the trees. This is shown in a histogram format.
Source tree size summary:

```
         num taxa
        4   |======================================== (274)
        5   |============================ (210)
        6   |================ (119)
        7   |========== (75)
        8   |======== (59)
        9   |=== (27)
        10  |== (16)
```

Clann then returns to the prompt for the next command.

Note:

It is necessary to run the "execute" command before any other, as all the other commands require that there are trees in memory.

alltrees command:

Purpose:

This command starts an exhaustive search of tree-space for the best supertree. It is advisable not to use an exhaustive search of tree-space when there are more than 10 taxa in the dataset, this is due to the prohibitively large number of trees it would be necessary to assess. Users should always check the number of trees in supertree space (from the input file summary) before using this command.

Options:

The following options are generally available with this command:

all

- Search all of tree-space (This is the default and it is not usually necessary to specify it specifically)

range <treenumber>-<treenumber>

- Search treespace in the area between tree number x and y. For example if there were 10 taxa in the supertree then supertree space would contain 2,027,025 trees. This could take a prohibitively long time on a single computer, but this option allows the user to get different computers to search different parts of tree space. i.e.

  "range 1-1000000" or *"range 1000001-2027025"*

savetrees <filename>

- By default, the supertree with the best score from an alltrees search will be written to the file "top_alltrees.txt". This file is overwritten each time an alltrees search is carried out. This option allows the user to specify a file name to contain the result.

create

- This option will write all the trees in supertree space along with their scores to the file "alltrees.txt". This command should be used with caution, as with large numbers of taxa, the resulting file can be very large.

weight

- This option specifies the weight used to normalise the scoring system for large (or small) tree bias. Depending on the optimality criteria used (see "set" command) there are different weighting schemes.

- Using the dfit optimality criterion, the options are:
  equal or **comparisons**
  equal applies no normalisation
  comparisons normalises so every tree has the same vote (regardless of tree size).

- Using the qfit optimality criterion, the options are:
  equal or **taxa** or **quartets**
  equal applies no normalisation
  taxa applies a normalisation that is calculated as n-3
  quartets normalises so every tree has the same vote (regardless of tree size).

- Using the sfit optimality criterion, the options are:
  equal or **splits**
  equal applies no normalisation
  splits normalises so every tree has the same vote (regardless of tree size).

Usage:

"alltrees create weight=equal savetrees=mytrees.txt"

Result:

Using whichever optimality criterion is applied, every tree in supertree-space is assessed. The best tree is identified and displayed to screen along with its score (the smaller the score the better the tree).

hs command:

Purpose:

This command performs a heuristic search of supertree space for the best tree. This is much quicker than an exhaustive search and should always be the method of choice for finding the best supertree, unless the number of taxa in the dataset is less than 10.

Options:

The following options are generally available with this command.

sample <integer number>

This option determines the number of randomly chosen supertrees are evaluated before heuristic searches are carried out. The top x number of trees are kept as starting points for each of the x repetitions of the heuristic search. The default value is 10,000.

swap <nni | spr>

- This option determines the type of heuristic search that is to be carried out.
  nni specifies that the nearest neighbor interchange method is to be used
  spr specifies that the sub tree pruning and regrafting method is to be used.

nsteps <integer number>

- This option specifies the maximum number of steps away from its original position that any branch may be swapped or regrafted. The default value is 5.

start <nj | random | filename>

- This option determines the starting point for the heuristic search.
- By default, the heuristic search starts with a neighbor-joining tree calculated from average consensus distances (with missing distances estimated using either ultrametric or 4-point condition distances). If more than one repetition of the heuristic search is to be carried out, the first repetition uses the nj tree as a starting point and every subsequent repetition uses the nj tree after a randomly chosen number of SPR operations having been carried out on it.
- If the user decides to use a randomly chsen tree as a starting point for the heuristic search, a certain number of random trees are evaluated (specified by the option 'sample') and the top few are retained (the number of which is specified by the option 'nreps'). These top trees are then used as starting points for each of the repetitions of the heuristic search.
- Finally, it is possible to specify a filename that contains trees which are to be used as starting points *If any other word other than 'nj' or 'random' is placed after the option "start", it is assumed that this is the name of a file containing starting trees.* ***This overrides the options "sample" and "nreps" .***
  i.e. *"start=random"* or *"start=mytrees.txt"*

nreps <integer number>

- This option specifies the number of repetitions of the heuristic search that are to be carried out. The best tree found is picked from all the repetitions carried out.

  i.e. *"nreps=10"*

maxswaps <integer number>

- This option specifies the maximum number of swaps/rearrangements to be tried during the search. This is set to 1,000,000 by default, but some searches may take too long so it may be desirable to set a lower limit. **The number of random samples carried out before the heuristic searches are not counted for the maxswaps limitation.**

  i.e. *"maxswaps=10000"*

savetrees <filename>

- By default, the supertree with the best score from an alltrees search will be written to the file "Heuristic_result.txt". This file is overwritten each time an alltrees search is carried out. This option allows the user to specify a file name to contain the result.

weight

- This option specifies the weight used to normalise the scoring system for large (or small) tree bias. Depending on the optimality criteria used (see "set" command) there are different weighting schemes.

- Using the dfit optimality criterion, the options are:
  equal or **comparisons**
  equal applies no normalisation
  comparisons normalises so every tree has the same vote (regardless of tree size).

- Using the qfit optimality criterion, the options are:
  equal or **taxa** or **quartets**
  equal applies no normalisation
  taxa applies a normalisation that is calculated as n-3
  quartets normalises so every tree has the same vote (regardless of tree size).

- Using the sfit optimality criterion, the options are:
  equal or **splits**
  equal applies no normalisation
  splits normalises so every tree has the same vote (regardless of tree size).

drawhistogram yes | no

- By default, this option is set to 'no'. but if set to yes, it will display in a histogram format the scores of the source trees compared to the best supertree found. Those source trees with a score of zero are completely compatible with the best supertree found.

nbins <integer number>

- By default, this value is set to 20. This is only used if the 'drawhistogram' option has been set to 'yes'. This sets the number of bins to be used when drawing the histogram.

histogramfile <file name>

- By default, this option is set to 'Heuristic_histogram.txt'. This is only used if the 'drawhistogram' option has been set to 'yes'. This specifies the file to which the values from the histogram are to be saved.

Options only available in MRP Criterion

The available options for Heuristic searches are different under the MRP criterion:

analysis <parsimony | nj>

These options allow the choice of carrying out a full parsimony heuristic search or instead carry out a neighbor-joining tree from the MRP matrix. The Neighbor joining tree could be used as a starting point for heuristic searches using other optimality criteria. The default is Parsimony.

swap <nni | spr | tbr>

This specifies the type of heuristic search to carry out in PAUP* on the MRP matrix. The default is TBR (tree bisection and reconnection).

addseq <simple | closest | asis | random | furthest>

This specifies the PAUP* settings for adding sequences during the parsimony heuristic search.

Options only available in the average consensus (avcon) Criterion

missing 4point | ultrametric

This specifies the method to use to estimate the missing data in the average consensus analysis. By default this is set to 4point which is the 4-point condition estimate (*34*).

Usage: *"hs swap=spr start=random nreps=10"*

Result:

Using whichever optimality criterion is chosen, the best tree found from the search(es) is displayed to screen and written to file (as described above). The score of the tree is also displayed. The lower the score the better the supertree.

usertrees command:

Purpose:

This command is designed to allow the user to assess which of a set of user-defined trees best represented the source trees (is the best supertree). The trees that the user needs to be checked must be contained in a file in phylip format.

Usage: **usertrees <filename>**

Options:

outfile <filename>

- By default the output filename is "Usertrees_result.txt". However, the user may specify another filename using this option.

weight

- This option specifies the weight used to normalise the scoring system for large (or small) tree bias. Depending on the optimality criteria used (see "set" command) there are different weighting schemes.

- Using the dfit optimality criterion, the options are:
  equal or **comparisons**
  equal applies no normalisation
  comparisons normalises so every tree has the same vote (regardless of tree size).

- Using the qfit optimality criterion, the options are:
  equal or **taxa** or **quartets**
  equal applies no normalisation
  taxa applies a normalisation that is calculated as n-3
  quartets normalises so every tree has the same vote (regardless of tree size).

- Using the sfit optimality criterion, the options are:
  equal or **splits**
  equal applies no normalisation
  splits normalises so every tree has the same vote (regardless of tree size).

Usage:

"*usertrees intrees.txt weight=equal outfile=mytrees.txt*"

Result:

Using whichever optimality criterion is applied, every tree in the input file is assessed. The best tree is identified and displayed to screen along with its score (the smaller the score the better the tree).

bootstrap (or boot) command:

Purpose:

This command performs a bootstrap analysis of the data. The bootstrapping is performed by resampling the trees (with replacement) from the input file and then performing a heuristic (or exhaustive) search of treespace to find the supertree it best represents. This is repeated an number of times (usually 100 times).

Options:

nreps <integer number>

- This option specifies the number of bootstrap replicates that are to be carried out. By default, this is set to 100.

  i.e. *"nreps=100"*

hsreps <integer number>

This option defines how many repetitions of the heuristic search are to be carried out on each bootstrap replicate. The default is 10.

sample <integer number>

This option determines the number of randomly chosen supertrees are evaluated before heuristic searches are carried out. The top x number of trees are kept as starting points for each of the x repetitions of the heuristic search. The default value is 10,000.

swap <nni | spr | all>

- This option determines the type of heuristic search that is to be carried out.
- *nni* specifies that the nearest neighbor interchange method is to be used

- *spr* specifies that the sub tree pruning and regrafting method is to be used.
- *all* specifies that an exhaustive search of supertree-space is to be used rather than a heuristic search to find the best tree for each replicate. *Note: "all" should not be used with more than 10 taxa due to the large number of trees in treespace.*

By default, this option is set to spr.

start <nj | random | filename>

- This option determines the starting point for the heuristic search.
- By default, the heuristic search starts with a neighbor-joining tree calculated from average consensus distances (with missing distances estimated using either ultrametric or 4-point condition distances). If more than one repetition of the heuristic search is to be carried out, the first repetition uses the nj tree as a starting point and every subsequent repetition uses the nj tree after a randomly chosen number of SPR operations having been carried out on it.
- If the user decides to use a randomly chsen tree as a starting point for the heuristic search, a certain number of random trees are evaluated (specified by the option 'sample') and the top few are retained (the number of which is specified by the option 'nreps'). These top trees are then used as starting points for each of the repetitions of the heuristic search.
- Finally, it is possible to specify a filename that contains trees which are to be used as starting points

If any other word other than 'nj' or 'random' is placed after the option "start", it is assumed that this is the name of a file containing starting trees. ***This overrides the options "sample" and "nreps" .***

i.e. *"start=random"* or *"start=mytrees.txt"*

nsteps <integer number>

- This option specifies the maximum number of steps away from its original position that any branch may be swapped or regrafted. The default value is 5.

treefile <filename>

- By default, the supertree(s) with the best score from each bootstrap replicate is written to the file "bootstrap.txt". This file is overwritten each time an bootstrap search is carried out. This option allows the user to specify different file name to the default.

maxswaps <integer number>

- This option specifies the maximum number of swaps/rearrangements to be tried during the search. This is set to 1,000,000 by default, but some searches may take too long so it may be desirable to set a lower limit. **The number of random samples carried out before the heuristic searches are not counted for the maxswaps limitation.**

  i.e. *"maxswaps=10000"*

weight

- This option specifies the weight used to normalise the scoring system for large (or small) tree bias. Depending on the optimality criteria used (see "set" command) there are different weighting schemes.

- Using the dfit optimality criterion, the options are:

equal or **comparisons**

equal applies no normalisation

comparisons normalises so every tree has the same vote (regardless of tree size).

- Using the qfit optimality criterion, the options are:

  equal or **taxa** or **quartets**

  equal applies no normalisation

  taxa applies a normalisation that is calculated as n-3

  quartets normalises so every tree has the same vote (regardless of tree size).

- Using the sfit optimality criterion, the options are:

  equal or **splits**

  equal applies no normalisation

  splits normalises so every tree has the same vote (regardless of tree size).

consensus strict | majrule | minor | <proportion>

By default Clann will perform a majority-rule consensus (with minor components) of the results of the bootstrap analysis. The option allows the user the type of consensus to be carried out.

'Strict' specifies that the consensus should only include relationships that appear 100% of the time in the bootstrapped trees.

'majrule' specieis that the consensus should only include relationships that appear great than 50% of the time in the bootstrapped trees.

'minor' specifies that that the consensus should only include relationships that appear great than 50% of the time in the bootstrapped trees, but that the

minor components that are compatible wth the majority-rule tree should also be included

Finally, the user has the option of specifying a proportion (greater than 0.5 and lessthan 1.0) that should be the minimum proportion that any relationship should appear in the bootstrapped trees before being included in the consensus analysis.

Usage*:*

"*bootstrap nreps=100 weight=equal outfile=mytrees.txt"*

or

"boot nreps=1000 weight=equal nsteps=10"

Result*:*

For each Bootstrap replicate, the best tree(s) found (using whatever criterion is set) is written to file (boostrap.txt by default). Where multiple trees are found as optimum for a bootstrap replicate, the trees for this replicate are down-weighted, by a value which is equal to 1/(number of optimum trees). This down-weighting is shown in the result file as a number in square brackets **before** the semi-colon (;). The score received by the trees for each replicate is displayed in square brackets **after** the semi-colon.

yaptp command:

Purpose:

This command means "Yet another permuted-tail-probability" test. This performs a randomisation test, where the signal in the source trees is destroyed and the data reanalysed to see how the score of the best tree compares to that from the un-permuted data. This test is generally carried out several times (usually 100 repetitions).

Options:

method <equiprobable | markovian>

This option specifies the method of randomizing the input trees. Ideally, an equiprobable approach is preferable, where, when a input tree is randomised, any other tree in tree-space with its leaf-set, is equally likely to be chosen. This however is not always possible, because this requires that we know the number of trees in tree-space in advance. This is difficult to calculate with large numbers of taxa. In this case a second approach (markovian) must be taken where we don't need to know the number of trees in supertree-space in order to create random trees. This has a disadvantage however in that this approach is biased towards palmate (balanced) versus pectinate (unbalanced) trees.

The default option is equiprobable, but it may be necessary to use a markovian approach with large numbers of taxa.

nreps <integer number>

- This option specifies the number of bootstrap replicates that are to be carried out. By default, this is set to 100.

i.e. *"nreps=100"*

hsreps <integer number>

This option defines how many repetitions of the heuristic search are to be carried out on each YAPTP replicate. The default is 10.

sample <integer number>

This option determines the number of randomly chosen supertrees are evaluated before heuristic searches are carried out. The top x number of trees are kept as starting points for each of the x repetitions of the heuristic search. The default value is 10,000.

search <nni | spr | all>

- This option determines the type of heuristic search that is to be carried out.
- *nni* specifies that the nearest neighbor interchange method is to be used
- *spr* specifies that the sub tree pruning and regrafting method is to be used.
- *all* specifies that an exhaustive search of supertree-space is to be used rather than a heuristic search to find the best tree for each replicate. *Note: "all" should not be used with more than 10 taxa due to the large number of trees in treespace.*

By default, this option is set to spr.

nsteps <integer number>

- This option specifies the maximum number of steps away from its original position that any branch may be swapped or regrafted. The default value is 5.

treefile <filename>

- By default, the supertree(s) with the best score from each bootstrap replicate is written to the file "yaptp.ph". This file is overwritten each time an

bootstrap search is carried out. This option allows the user to specify different file name to the default.

maxswaps <integer number>

- This option specifies the maximum number of swaps/rearrangements to be tried during the search. This is set to 1,000,000 by default, but some searches may take too long so it may be desirable to set a lower limit. **The number of random samples carried out before the heuristic searches are not counted for the maxswaps limitation.**
  i.e. *"maxswaps=10000"*

weight

- This option specifies the weight used to normalise the scoring system for large (or small) tree bias. Depending on the optimality criteria used (see "set" command) there are different weighting schemes.

- Using the dfit optimality criterion, the options are:
  equal or **comparisons**
  equal applies no normalisation
  comparisons normalises so every tree has the same vote (regardless of tree size).

- Using the qfit optimality criterion, the options are:
  equal or **taxa** or **quartets**
  equal applies no normalisation
  taxa applies a normalisation that is calculated as n-3
  quartets normalises so every tree has the same vote (regardless of tree size).

- Using the sfit optimality criterion, the options are:
  equal or **splits**

equal applies no normalisation

splits normalises so every tree has the same vote (regardless of tree size).

Usage*:*

"yaptp nreps=100 swap=spr weight=equal treefile=myresults.txt"

Result:

For each yaptp replicate, the best tree(s) found and their scores (using whatever criterion is set) is written to file (yaptp.ph by default). The purpose of this test is to assess the distribution of scores received when the data is randomised and to compare that to the scores from the real data. The scores received for each replicate is display in square brackets after the tree in the result file.

includetrees command:

Purpose:

This command allows the user to include source trees for subsequent analysis (that have been previously excluded). Using this command it is possible to include source trees from different categories (like "with a certain number of taxa" or "containing a certain taxon" etc…).

Options:

range <integer value> - <integer value>

- This option re-includes all the trees within the specified range for subsequent analyses.

size equalto <integer value>

lessthan <integer value>

greaterthan <integer value>

This option re-includes all previously excluded source trees that contain the specified number of taxa.

The option **'equalto'** specifies that the trees to be included must have that specified number of taxa.

The option **'lessthan'** specifies that the trees to be included must have less than the specified number of taxa.

The option '**greaterthan**' specifies that the trees to be included must have more than the specified number of taxa.

namecontains <character string>

This option re-includes all previously excluded source trees that contain the string specified in their name. The string may be a subset of the name of the tree or identical to the name.

containstaxa <character string>

This option re-includes only those previously excluded source trees that contain the taxa specified. A substring of the taxa name may be specified. Only one character string can be used per usage of the option 'containstaxa' but the option 'containstaxa' may be used more than once per command.

score <min score> - <max score>

This option re-includes all source trees that have a score within the range specified. This option requires that there is a supertree in memory.

Usage:

"includetrees range=10-12"

or

"includetrees size lessthan=5"

or

"includetrees containstaxa=Arabadopsis"

Result:

The result of this command, re-includes all the source trees specified by the options used in the command. This command does not include anything exclusively, it only adds these certain sets of source trees to those to be subsequently analysed. To include anything exclusively, it is necessary to exclude all the source trees first and then re-include those required for the analysis.

excludetrees command:

Purpose:

This command allows the user to exclude source trees for subsequent analysis. Using this command it is possible to exclude source trees from different categories (like "with a certain number of taxa" or "containing a certain taxon" etc…).

Options:

range <integer value> - <integer value>

- This option excludes all the trees within the specified range for subsequent analyses.

size equalto <integer value>

lessthan <integer value>

greaterthan <integer value>

This option excludes all source trees that contain the specified number of taxa.

The option **'equalto'** specifies that the trees to be excluded must have that specified number of taxa.

The option **'lessthan'** specifies that the trees to be excluded must have less than the specified number of taxa.

The option '**greaterthan**' specifies that the trees to be excluded must have more than the specified number of taxa.

namecontains <character string>

This option excludes source trees that contain the string specified in their name. The string may be a subset of the name of the tree or identical to the name.

containstaxa <character string>

This option excludes only those source trees that contain the taxa specified. A substring of the taxa name may be specified. Only one character string can be used per usage of the option 'containstaxa' but the option 'containstaxa' may be used more than once per command.

score <min score> - <max score>

This option excludes all source trees that have a score within the range specified. This option requires that there is a supertree in memory.

Usage:

"excludetrees range=10-12"

or

"excludetrees size lessthan=5"

or

"excludetrees containstaxa=Arabadopsis"

Result:

The result of this command, excludes all the source trees specified by the options used in the command. This command does not exclude anything exclusively, it only adds these source trees to whatever has been excluded previously. To exclude anything exclusively, it is necessary to include all the source trees first and then exclude those required for the analysis.

showtrees command:

Purpose:

This command allows the user to display source trees to the screen or count the number of sourcetrees, which have certain properties. (like "with a certain number of taxa" or "containing a certain taxon" etc…).

Options:

range <integer value> - <integer value>

- This option shows all the trees within the specified range.

size equalto <integer value>

lessthan <integer value>

greaterthan <integer value>

This option shows all source trees that contain the specified number of taxa.

The option **'equalto'** specifies that the trees to be shown must have that specified number of taxa.

The option **'lessthan'** specifies that the trees to be shown must have less than the specified number of taxa.

The option '**greaterthan**' specifies that the trees to be shown must have more than the specified number of taxa.

namecontains <character string>

This option shows source trees that contain the string specified in their name. The string may be a subset of the name of the tree or identical to the name.

containstaxa <character string>

This option shows only those source trees that contain the taxa specified. A substring of the taxa name may be specified. Only one character string can be used per usage of the option 'containstaxa' but the option 'containstaxa' may be used more than once per command.

score <min score> - <max score>

This option shows all source trees that have a score within the range specified. This option requires that there is a supertree in memory.

savetrees yes | no

This option specifies whether or not the trees displayed to the screen are to bo saved to file. This allows the user to create files containing subsets of the original dataset according to the criteria specified.

filename <output filename>

If the user has specified that the trees shown are to be saved to file, then this command allows the user to specify the file to which they are saved. By default the file is called 'showtrees.txt'.

display yes | no

This option specifies whether or not the trees that meet the conditions are to be displayed individually to the screen. If this is set to 'no' then only a total of the number of trees that meet the criteria specified is displayed. By default, this is set to 'yes'.

Usage:

"showtrees range=10-12"

or

"showtrees size lessthan=5"

or

"showtrees containstaxa=Arabadopsis"

Result:

The result of this command, shows all the source trees that meet the criteria specified by the options. This command works in conjunction with the includetrees and excludetrees command so that trees that have been excluded from the analysis are not displayed in the showtrees command.

consensus command:

Purpose:

This command is used construct a consensus tree for all the universally distributed source trees (all the source trees that contain all the taxa).

Options:

method strict | majrule | minor | <proportion>

By default Clann will perform a majority-rule consensus (with minor components) of the universally distributed trees. This option allows the user to specify the type of consensus to be carried out.

'Strict' specifies that the consensus should only include relationships that appear 100% of the time in the bootstrapped trees.

'majrule' specieis that the consensus should only include relationships that appear great than 50% of the time in the bootstrapped trees.

'minor' specifies that that the consensus should only include relationships that appear great than 50% of the time in the universally distributed trees, but that the minor components that are compatible with the majority-rule tree should also be included.

Finally, the user has the option of specifying a proportion (greater than 0.5 and lessthan 1.0) that should be the minimum proportion that any relationship should appear in the universally

distributed trees before being included in the consensus analysis.

filename <output file name>

This option allows the user to choose the name of the file to which the consensus tree is to be saved. By default this is set to 'consensus.ph'

Usage:

"consensus method=majrule"

or

"consensus method=.7 filename=myconsensus.txt"

Result:

This command will calculate the consensus of the universally distributed trees source trees. The output will look as follows:

```
Consensus settings:
        Consensus of 17 universally distributed source trees
        Only relationships with 50% support or greater (including
congruent minor components) are included in the consensus
        Consensus file = consensus.ph



Sets included in the consensus tree
        .********.******        1.00
        .......*...*....        1.00
        ...........**..         0.94
        ..**............        0.94



Minor Components included in the consensus tree
        .********.****.*        0.41
        .***..*.........        0.35
        .***............        0.29
        *........**...*.        0.12
        .......**..*....        0.06
        *****.*********.        0.06
        ****..******..*.        0.06
        ....*.......**..        0.06
```

```
Sets not included in the consensus tree
        ......*.*.......        0.12
        .*....*.........        0.06
        *........*....**        0.06
        .********..*****        0.06
        .****............        0.06
        .*..*............        0.06
        *...****.*******        0.06
        .***..*.*.......        0.06
        ..**..*.........        0.06
        **..**.*.*******        0.06


                                          +------------------- Taxon_A
                                +---------| 1.00
                                |         +------------------- Taxon_B
                      +---------| 0.12
                      |         |----------------------------- Taxon_C
                      |         |
                      |         +----------------------------- Taxon_D
                      |
                      |                   +------------------- Taxon_E
                      |         +---------| 0.29
                      |         |         |          +--------- Taxon_F
             +--------|---------| 0.35    +----------| 0.94
             |        |         |                     +--------- Taxon_G
             |        |         |
             |        |         +----------------------------- Taxon_H
             |        |
             |        |                   +------------------- Taxon_I
    +--------| 0.06    |         +---------| 1.00
    |        |         +---------| 0.06    +------------------- Taxon_J
    |        |                   |
    |        |                   +----------------------------- Taxon_K
    |        |
    |        |         +------------------------------------- Taxon_L
    |        +---------| 0.06
    |                  |         +----------------------------- Taxon_M
    |                  +---------| 0.94
    |                            +----------------------------- Taxon_N
    |
    |        +-------------------------------------------------- Taxon_O
    +--------| 0.41
             +-------------------------------------------------- Taxon_P


clann>
```

In this output, the number of universally distributed trees in the dataset are displayed and the different relationships along with the number of times they were found are displayed to screen. In this summary each line of dots '.' Or stars '*' represent a single split

found in the dataset. The taxa are in the same order as they were displayed in the input file summary. All the taxa with a dot under them are found on one side of the split and all the taxa with a star were found on the other side of the split. The numbers represent the proportion of universally distributed input trees that contained that particular split.

Based on the type of consensus specified by the user, the relationships are grouped into three categories:

- Sets included in the consensus tree
- Minor Components included in the consensus tree
- Sets not included in the consensus tree

The minor components, if included, are those that are compatible with the major components included in the consensus tree.

Finally Clann displays the tree in Graphical ascii format along with the level of support found for each internal branch.

deletetaxa command:

Purpose:

This command is used to delete particular taxa from the dataset. The taxa is pruned from each source tree that contains it. If any trees have less than 4 taxa after the taxon is pruned they are removed from the dataset. This is a permanent action, once a taxon has been deleted it cannot be re-included without re-loading the original dataset.

Usage:

deletetaxa <taxa name> <taxa name> <taxa name> ….

User may specify as many taxa names as desired after the command 'deletetaxa'. Substrings of the taxon names may also be used.

For instance:

'deletetaxa taxon_A taxon_E'

or

'deletetaxa Arabidop'

generatetrees command:

Purpose:

This command is used create random supertrees and assess them against the source tree dataset in memory. This allows the user to assess the shape of supertree space.

Options:

method equiprobable | markovian

- This option sets the method to be used to construct the random supertrees. The most desirable method is 'equiprobable' as this gives any tree in supertree space an equal chance of being chosen. This however is not always possible to use as it requires the knowledge of how many supertrees there are in supertree space. This becomes difficult to calculate when the number of taxa becomes large. By default the generatetrees command will use the equiprobable method.
- The 'markovian' method of creating random supertrees does not need to have any knowledge of the size of supertree space. It uses a random-star decomposition method of constructing the random tree. This makes it equally likely for a palmate relationship to be included in the tree as a pectinate one. However, this does not reflect supertree space, where there are far more pectinate relationships than palmate ones. This means that the markovian method will create proportionally more palmate (balanced) trees than it should.

ntrees all | <integer number>

This option sets the number of random supertrees to be assessed. By default this is set to 100. The user may specify 'all', this will make clann try all possible supertrees, which may take some time if the number of taxa is large.

nbins <integer number>

This option sets the number of bins into which the scores of the random supertrees are to be organized for the histogram summary of the results. By default this is set to 20.

outfile <output file>

This option sets the name of the output file to which the results of the analysis are to be written. By default this is set to 'histogram.txt'.

sourcedata real | randomised | ideal

This option sets the kind of data to which, the random supertrees are to be compared.

'real' specifies that the random supertrees are to be compared to the real source data. This is the default setting.

'randomised' specifies that the supertrees are to be compared to a randomised version of the source trees. In this analysis, each of the source trees are randomised so that any congruent signal between them should be broken. This allows the comparison of the distribution of supertrees to randomised data to that of real data.

'ideal' specifies that the supertrees are to be compared to an 'idealised' version of the real dataset. This means that each of the source trees are changed so that they are completely compatible with a 'best' supertree calculated elsewhere. This allows the user to compare the distribution of 'ideal' data to the 'real' data.

savescores yes | no

This option sets whether or not all the score of the random supertrees are to be saved to file. By default this is set to 'no'. If this is set to 'yes' all the scores of the random supertrees are saved to the file '*allscores.txt'*.

supertree memory | <supertree file name>

This option is only valid when "sourcedata" is set to "ideal". This specifies the location of the supertree to use when creating the ideal dataset. By default this is set to "memory" and if there is no supertree in memory, it will return an error.

savesourcetrees yes | no

This option specifies whether or not the source trees used during the analysis are to be saved to file. This allows the user to save an 'idealised' or 'randomised' version of their data to file. By default this is set to 'no'. If this option is set to 'yes' the trees are saved to the file called '*sourcetrees.ph*'.

Usage:

'generatetrees ntrees=1000 sourcedata=ideal savesourcetrees=yes'

or

'generatetrees'

Results:
The following is an example of the results from 'generatetrees'

```
Results as follows:


395.06 - 397.02 | (2)
397.03 - 398.99 |=== (7)
399.00 - 400.96 |===== (11)
400.97 - 402.93 |============ (25)
402.94 - 404.90 |============== (33)
404.91 - 406.87 |========================== (54)
406.88 - 408.84 |====================== (47)
408.85 - 410.81 |================================ (69)
410.82 - 412.78 |=================================== (77)
412.79 - 414.75 |================================ (74)
414.76 - 416.72 |=================================== (76)
416.73 - 418.69 |======================================== (83)
418.70 - 420.67 |================================ (74)
420.68 - 422.64 |================================ (73)
422.65 - 424.61 |============================== (65)
424.62 - 426.58 |======================================= (82)
426.59 - 428.55 |=================== (43)
428.56 - 430.52 |================= (38)
430.53 - 432.49 |========== (23)
432.50 - 434.46 |==================== (44)



Moments of the Distribution:

        Mean = 417.344067
        Variance = 74.647698
        Standard Deviation = 8.639890
        Skewness = 0.023036
        Standard deviation of skewness = 0.077460
```

The two columns on the left represent the range of score of the supertrees, the histogram represents the number of trees having a score that falls in that range. At the bottom are some simple statistics about the distribution. These are only descriptive statistics and hold no statistical significance.

nj command:

Purpose:

This command is used calculate a neighbor-joining tree from average consensus distances. Missing data are estimated using either a 4-point condition or ultrametric distances.

Options:

missing 4point | ultrametric

By default this is set to 4point. This is the more robust of the two methods but needs more information than the ultrametirc method.

savetrees <file name>

This command saves the neighbor-joining tree to file. By default the tree is saved to the file 'NJtree.ph'.

Usage:

'nj'

or

'nj missing=ultrametric savetrees=mytree.ph'

rfdists command:

Purpose:

This command is used calculate robinson-foulds (sometimes called symmetric) distances between the source trees. For each comparison between two source trees, the two trees are pruned down to their shared taxa and (if they share more than 3 taxa) a robinson-fould distance is calculated.

Options:

filename <output file name>

This option sets name of the file to which the calculated distances are to be written. By default this is set to 'robinson_foulds.txt'.

output matrix | vector

This option sets the output style. By default this is set to matrix.

missing none | 4point | ultrametric

This option allows the user to calculate estimates for comparisons between trees which were incomparable because there was not enough taxon overlap. By default this is set to 'none' as calculating this may not make any sense. The two methods here to calculate missing values are '4point' [4-point condition estimates] or ultrametric [from the ultrametric distances].

set command:

Purpose:

This command is used to set global user defined options. The options set using this command are valid for all analyses.

Options:

criterion < dfit | sfit | qfit | mrp >

- This option sets the optimality criterion to be used to search supertree space. Presently there are 4 optimality criteria implemented:
- dfit is the optimality criterion called the most similar supertree method (Creevey et al 2004). This is the default for this option.
- sfit is the optimality criterion called the maximum splits (or components) fit.
- qfit is the optimality criterion called the maximum quartet fit.
- mrp is the optimality criterion known as matrix representation using parsimony. Note clann uses Paup* (*35*) to carry out the parsimony step of the MRP analysis. It is necessary to have Paup* (*35*) installed for this purpose.

Usage:

"set criterion=sfit"

or

"set criterion=dfit"

! command:

Purpose:

The purpose of this command is to return the user to a c-shell on the operating system, while keeping clann running in the background. This allows the user to check the contents of result files or run other programs without effecting clann. To return to Clann, type "exit" at the c-shell prompt.

Usage:

"!"

or

"!"

quit command:

Purpose:

To quit Clann and return to the operating system.

Usage:

"quit"

Tutorial:

This short tutorial will bring the user step by step through the process of analysing a dataset using different optimality criteria and methods of investigating the data. For the purposes of this tutorial, a file called "tutorial_trees.ph" has been included with the distribution of the software. Please note that these are only suggestions as to the commands to use when analysing data, there is no excuse for just following a "recipe" and not thinking about your data.

1) There are two ways of reading in a file of source trees into Clann. The first is to use the "exe" command at the "clann>" prompt. For example:

   "clann>exe tutorial_trees.ph"

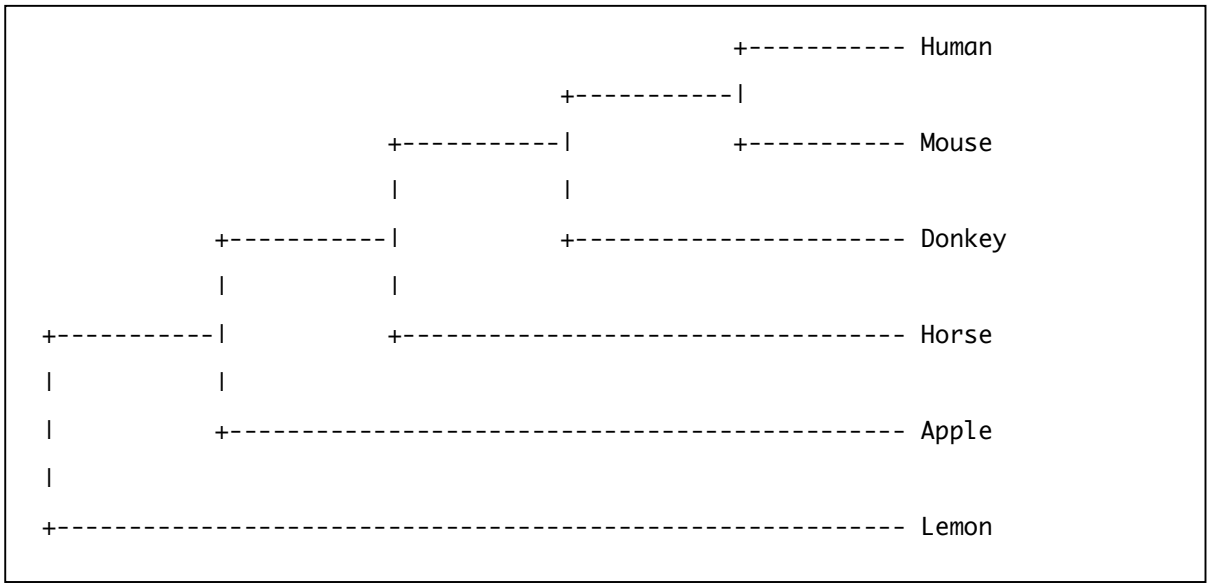
   The second way is to place the name of the file as an argument when you start Clann. For example:

   "%> clann tutorial_trees.ph"

   This starts Clann and immediately executes the source trees. The after the execution of the source trees file a summary of the number of trees etc (as described in the section "exe command") is displayed to the screen.

```
Source tree summmary:

        ------------------------------------------------
                Number of input trees: 22
                Number of unique taxa: 6
                Total unrooted trees in Supertree space?
                        105

        Occurrence summary:

                number  Taxa name           Occurrence

                0       Human               22
                1       Mouse               19
                2       Apple               19
                3       Horse               21
                4       Lemon               19
                5       Donkey              18


        Co-occurrence summary:

                Taxa Number

                      0     1     2     3     4     5
                0     -
                1     19    -
                2     19    16    -
                3     21    19    18    -
                4     19    16    17    18    -
                5     18    15    17    17    17    -


        Source tree size summary:

        num taxa
        4   |================= (6)
        5   |===== (2)
        6   |======================================== (14)
```

2) Next it is necessary to decide which optimality criteria is to be used to find the best supertree. By default Clann will use the dfit (most similar supertree method) criterion, and we will use this.
3) To carry out a quick analysis type ‘nj’ to get a quick neighbor-joining tree. The result will look something like this:

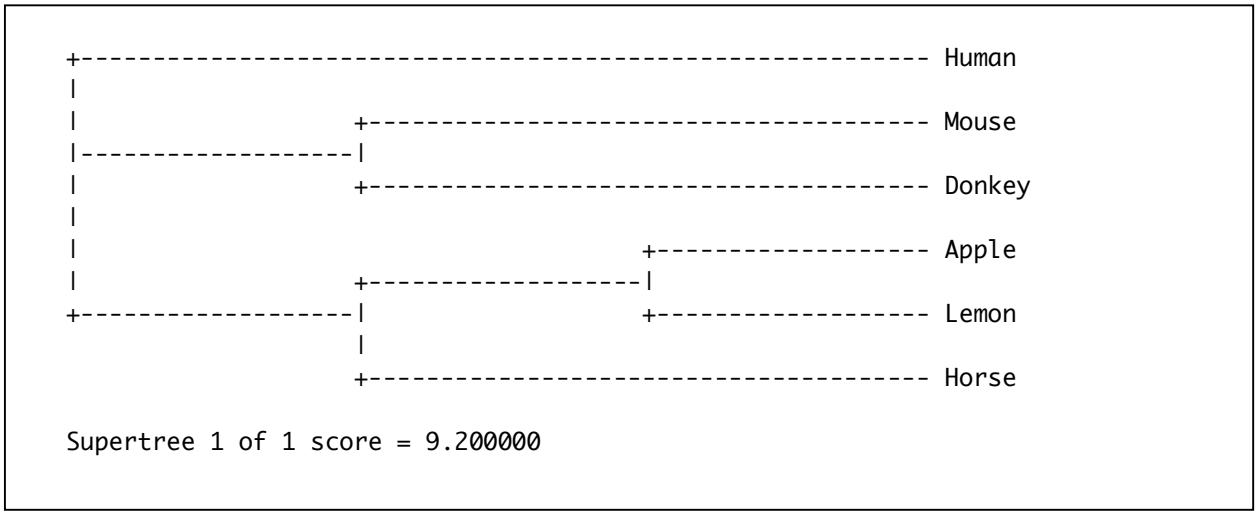


This is a quick tree and the results can be misleading. Immediately we can see that the Donkey and Horse are not grouping together.

4) Next perform a heuristic search of supertree space to get a better idea of the 'true' tree,
   type:

   "hs"

   The result of the analysis looks like this:



This is the best tree that Clann found with the DFIT criterion. Immediately we can see that there is something wrong, as the

Donkey seems to be grouping with the Mouse to the exclusion of the Humans.

Ideally we would have liked the donkey and the Horse to group together, but there is something in our data telling us differently.

5) From the source tree size summary we can see that there are 14 trees that are universally distributed (have all 6 taxa). So we can carry out a consensus of those trees: Type 'consensus' and the following is the result:

```
Consensus of 14 universally distributed source trees
        Only relationships with 50% support or greater (including
congruent minor components) are included in the consensus
        Consensus file = consensus.ph



Sets included in the consensus tree
        ..*.*.  0.71
        **...*  0.57
        **....  0.50



Minor Components included in the consensus tree


Sets not included in the consensus tree
        ..*.**  0.36
        .*...*  0.29
        *....*  0.21
        ....**  0.21
        ...*.*  0.07
        ..*..*  0.07


                                      +------------------- Human
                   +------------------| 0.50
+------------------| 0.57             +------------------- Mouse
|                  |
|                  +--------------------------------------- Donkey
|
|                  +--------------------------------------- Apple
|------------------| 0.71
|                  +--------------------------------------- Lemon
|
+------------------------------------------------------------ Horse
```

The consensus of the universally distributed trees seems to agree with the neighbor-joining tree. Next lets try to assess the level of support across all the source trees by bootstrapping:

6) To perform a Bootstrap analysis, simply type the following:

"boot"

If further options are required (like the type of search to be used and the number of bootstrap repetitions) type something like the following:

"boot nreps=10"

The results will look something like this:

```
Sets included in the consensus tree
        **.*.*  0.80
        **...*  0.70
        **....  0.60



Minor Components included in the consensus tree


Sets not included in the consensus tree
        .*...*  0.40
        **.*..  0.30
        ****..  0.20


                                      +------------------- Human
                    +-----------------| 0.60
+-------------------| 0.70            +------------------- Mouse
|                   |
|                   +--------------------------------------- Donkey
|
|                   +--------------------------------------- Apple
|-------------------| 0.80
|                   +--------------------------------------- Lemon
|
+------------------------------------------------------------ Horse
```

We find that there is reasonable support for the different relationships in the tree, but to our eye we think that the position of Donkey and Horse is wrong. Also the Bootstrap support doesn't seem very high. The next test we could try is to test for signal

within our data. Although with bootstrap support as high as we found it is unlikely that the data will fail this test. To run a yaptp test, type ‘yaptp’. The results are as follows:

```
14.13 - 14.55   |============== (6)
14.56 - 14.98   |========= (4)
14.99 - 15.40   |======= (3)
15.41 - 15.83   |============================== (14)
15.84 - 16.26   |================================ (15)
16.27 - 16.68   |================================== (16)
16.69 - 17.11   |==================================== (17)
17.12 - 17.54   |======================== (11)
17.55 - 17.96   |======================== (11)
17.97 - 18.39   |========= (4)



Moments of the Distribution:

        Mean = 16.273928
        Variance = 3.957324
        Standard Deviation = 1.989302
        Skewness = -6.455708
        Standard deviation of skewness = 0.243733
```

These are the results of 100 repetitions of the YAPTP test. The histogram represents the distribution of the scores of the best trees found for each repetition. We find that the best score we found was 14.13 and from the earlier heuristic search we know that the score of the best tree from the real data is 9.2. This data passes the YAPTP test because the score of the best tree is better than any of the repetitions from the YAPTP test.

7) Next we will try to examine the individual source trees to look for anomalies. Type ‘showtrees’ and all the source trees will display to the screen like this:

```
Tree number 3
Tree name =  fast 3
Weight = 1.000000
Score = 0.800000


                                    +------------------- Human
                    +---------------|
+-------------------|                +------------------- Donkey
|                   |
|                   +--------------------------------------- Mouse
|
|                   +--------------------------------------- Apple
|-------------------|
|                   +--------------------------------------- Lemon
|
+------------------------------------------------------------ Horse



Tree number 4
Tree name =  slow 1
Weight = 1.000000
Score = 0.800000


+------------------------------------------------------------ Human
|
|                   +--------------------------------------- Apple
|-------------------|
|                   |                +------------------- Donkey
|                   +----------------|
|                                    +------------------- Lemon
|
+------------------------------------------------------------ Horse
```

Each tree is drawn to screen along with its number, name, weight and score. The name of the tree is defined by the user in the input file [open the input file to see how that's done]. It can be useful to use the name to describe something about the data underlying the tree. For instance in this case all the trees were either labeled "fast" or "slow", representing whether the gene in question was thought to be fast or slowly evolving. We can use the names to just analyse those genes that are evolving either fast or slowly. (the same commands can be used to single out subsets of the dataset based on many different criteria).

First we will exclude the fastlly evolving genes and only analyse the slowly evolving ones. Type 'excludetrees namecontains=fast'.

Next Re-analyse the dataset using a Heuristic search and consensus analysis. The result is still not very well supported.

Re-include the fast evolving trees and exclude the slowly evolving trees by typing the commands:

'includetrees namecontains=fast'
then
'*excludetrees namecontains=slow'*

This doesn't seem to solve the problem either. There must be something else about the data that is causing the low support.
Re-include the slowly evolving genes and re-examine the data using the showtrees command. Look at the positions of the taxa with particular reference to the Horse and Donkey. Notice anything?

For instance:

```
Tree number 3
Tree name =  fast 3
Weight = 1.000000
Score = 0.000000

                                  +------------------- Human
                    +-------------|
+-------------------|             +------------------- Donkey
|                   |
|                   +------------------------------------ Mouse
|
|                   +------------------------------------ Apple
|-------------------|
|                   +------------------------------------ Lemon
|
+------------------------------------------------------------ Horse


Tree number 4
Tree name =  slow 1
Weight = 1.000000
Score = 1.000000


+------------------------------------------------------------ Human
|
|                   +------------------------------------ Apple
|-------------------|
|                   |             +------------------- Donkey
|                   +-------------|
|                                 +------------------- Lemon
|
+------------------------------------------------------------ Horse


Tree number 5
Tree name =  slow 2
Weight = 1.000000
Score = 0.000000


                    +------------------------------------ Human
+-------------------|
|                   |             +------------------- Mouse
|                   +-------------|
|                                 +------------------- Donkey
|
|                   +------------------------------------ Apple
|-------------------|
|                   +------------------------------------ Lemon
|
+------------------------------------------------------------ Horse
```

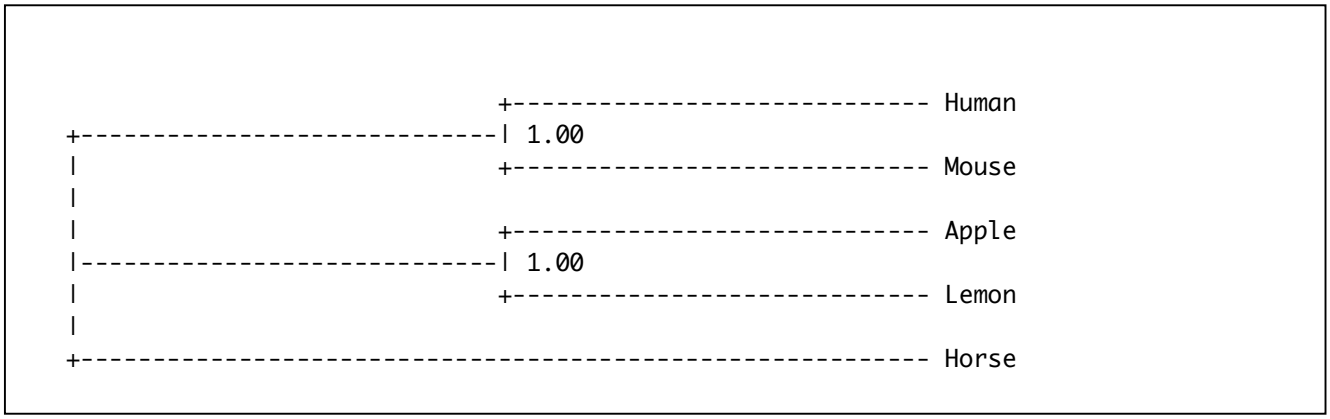
Here in these three trees we can see a problem with the taxon 'Donkey', In these three trees alone it is grouping with three different taxa, none of which are the Horse.

Upon examination of the rest of the trees you can see that the trend continues throughout the rest of the data.

The low support that has been received by the data may be because of the taxon 'Donkey' moving around in the dataset.

Lets try to delete the taxon 'Donkey' and check the results. To do this type: *'deletetaxa Donkey'*. Clann will ask you if you are sure you wish to continue as this is a permanent action. Type yes.

Now do a bootstrap analysis. The result will look like this:



We seem to have found the problem. Now try the other analyses we did earlier to see if it has improved their results too.

We would always recommend that users investigate their data as much as possible. Clann has been designed to make this as easy as possible for phylogenetic data.

References:

1. M. J. Sanderson, A. Purvis, C. Henze, *Trends in Ecology & Evolution* **13**, 105 (Mar, 1998).
2. V. Daubin, M. Gouy, G. Perriere, *Genome Research* **12**, 155 (2001).
3. F. Tekaia, A. Lazcano, B. Dujon, *Genome Research* **9**, 550 (Jun, 1999).
4. W. C. Lathe, B. Snel, P. Bork, *Trends in Biochemical Sciences* **25**, 474 (Oct, 2000).
5. Y. I. Wolf, I. B. Rogozin, N. V. Grishin, R. L. Tatusov, E. V. Koonin, *BMC Evolutionary Biology* **1** (2001).
6. B. Snel, P. Bork, M. A. Huynen, *Nature Genetics* **21**, 108 (Jan, 1999).
7. S. T. Fitz-Gibbon, C. H. House, *Nucleic Acids Research* **27**, 4218 (Nov 1, 1999).
8. C. H. House, S. T. Fitz-Gibbon, *Journal of Molecular Evolution* **54**, 539 (Apr, 2002).
9. J. J. Wernegreen, *Nature Reviews Genetics* **3**, 850 (2002).
10. A. L. Reysenbach, E. Shock, *Science* **296**, 1077 (2002).
11. A. L. Hughes, *Adaptive evolution of genes and genomes* (Oxford University Press, Oxford, 1999), pp. 288 p.: p., ill., 24 cm.
12. J. O. Korbel, B. Snel, M. A. Huynen, P. Bork, *Trends in Genetics* **18**, 158 (2002).
13. J. G. Lawrence, *Trends in Microbiology* **5**, 355 (1997).
14. N. V. Grishin, Y. I. Wolf, E. V. Koonin, *Genome Research* **10**, 991 (Jul, 2000).
15. Y. I. Wolf, I. B. Rogozin, N. V. Grishin, E. V. Koonin, *Trends in Genetics* **18**, 472 (2002).
16. S. A. Teichmann, G. Mitchison, *Journal of Molecular Evolution* **49**, 98 (Jul, 1999).
17. O. Matte-Tailliez, C. Brochier, P. Forterre, H. Philippe, *Molecular Biology and Evolution* **19**, 631 (2002).
18. C. Brochier, E. Bapteste, D. Moreira, H. Philippe, *Trends in Genetics* **18**, 1 (2003).
19. J. R. Brown, C. J. Douady, M. J. Italia, W. E. Marshall, M. J. Stanhope, *Nature Genetics* **28**, 281 (Jul, 2001).
20. S. Hansmann, W. Martin, *International Journal of Systematic and Evolutionary Microbiology* **50**, 1655 (2000).
21. B. R. Baum, *Taxon* **41**, 3 (Feb, 1992).
22. M. A. Ragan, *Biosystems* **28**, 47 (1992).
23. T. Sicheritz-Ponten, S. G. E. Andersson, *Nucleic Acids Research* **29**, 545 (Jan 15, 2001).
24. F.-J. Lapointe, G. Cucumel, *Syst Biol* **46**, 306 (1997).
25. O. Zhaxybayeva, J. P. Gogarten, *BMC Evolutionary Biology* **3** (2002).

26. W. F. Loomis, D. W. Smith, *Proceedings of the National Academy of Sciences of the United States of America* **87**, 9093 (Dec, 1990).
27. M. Steel, *Journal of Classification* **9**, 91 (1992).
28. O. R. P. Bininda-Emonds, J. L. Gittleman, A. Purvis, *Biological Reviews of the Cambridge Philosophical Society* **74**, 143 (May, 1999).
29. O. R. P. Bininda-Emonds, M. J. Sanderson, *Systematic Biology* **50**, 565 (2001).
30. N. Salamin, T. R. Hodkinson, V. Savolainen, *Systematic Biology* **51**, 136 (2002).
31. J. L. Thorley, M. Wilkinson, in *Bioconsensus. DIMACS Series in Discrete Mathemetics and Theoretical Computer Science* F. S. Roberts, Ed. (The American Mathematical Society, New York, 2003), vol. 61, pp. 185-194.
32. F.-J. Lapointe, G. Cucumel, *Systematic Biology* **46**, 306 (1997).
33. C. Levasseur, F. J. Lapointe, *Dimacs series in discrete mathematics and theoretical computer science* (2003).
34. F. J. Lapointe, C. Levasseur, in *Phylogenetic Supertrees: Combining information to reveal the Tree of Life* O. R. P. Bininda-Emonds, Ed. (Kluwer Academic, Dordrecht, 2004), vol. 4.
35. D. L. Swofford, *PAUP*. Phylogenetic Analysis Using Parsimony (*and other methods). Version 4* (Sinauer Associates, Sunderland, Massachusetts., 2002), pp.